In [1]:

```
################## Install Packages ###############
# !pip install catboost
# !pip install lightgbm
```

In [2]:

```
## Impoorting the required packages for
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn import linear_model

from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as pl
from sklearn.metrics import log_loss
import warnings
# warnings.filterwarnings('ignore')

from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import traceback

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier,\
RandomForestClassifier, BaggingClassifier, AdaBoostClassifier,\
GradientBoostingClassifier, HistGradientBoostingClassifier;
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.metrics import log_loss
from sklearn.ensemble import HistGradientBoostingClassifier

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import log_loss
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
/home/maheswari21303/.local/lib/python3.9/site-packages/xgboost/compat.py:31: FutureWarning: pandas.Int64Index is deprecate
d and will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [3]:

```
# to mount drive
# from google.colab import drive
# drive.mount('/content/drive')
```

In [4]:

```
####### Reading the file (after feature engineering)

cancer_df=pd.read_csv('final_trianing_frame.csv')
cancer_df
```

Out[4]:

	Class	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	loss	respons	su
0	1	0	0	0	0	0	0	0	0	0	...	0.006990	0.037058	0.(
1	2	0	0	0	0	0	0	0	0	0	...	0.012453	0.004001	0.(
2	2	0	0	0	0	0	0	0	0	0	...	0.012453	0.004001	0.(
3	3	0	0	0	0	0	0	0	0	0	...	0.025350	0.009774	0.(
4	4	0	0	0	0	0	0	0	0	0	...	0.011643	0.000000	0.(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8572	9	0	0	0	0	0	0	0	0	0	...	0.002705	0.019934	0.(
8573	9	0	0	0	0	0	0	0	0	0	...	0.010511	0.003670	0.(
8574	9	0	0	0	0	0	0	0	0	0	...	0.000000	0.013410	0.(
8575	9	0	0	0	0	0	0	0	0	0	...	0.017573	0.007047	0.1
8576	9	0	0	0	0	0	0	0	0	0	...	0.014435	0.003815	0.2

8577 rows × 3374 columns

In [ ]:

```

```

In [5]:

```
###### A single shot way to find out the relivant Models


classifiers = [     SVC(kernel='linear',probability = True),
                    SVC(degree=7, kernel='poly', probability = True),
                    SVC(kernel = 'rbf', probability = True),

                     LogisticRegression(multi_class='multinomial', solver='lbfgs'),

                     ExtraTreesClassifier(),
                     RandomForestClassifier(),
                     BaggingClassifier(),
                     AdaBoostClassifier(),
                     GradientBoostingClassifier(),
                     HistGradientBoostingClassifier(),
                     CatBoostClassifier(),
                     LGBMClassifier(),
                     XGBClassifier(),
                     MultinomialNB(),
                     KNeighborsClassifier(),

               ]

names = [           'Linear SVM',
                    'Polynomial SVM',
                    'RBF SVM',
                    'Logistic_Regression',
                    'Extra_Trees_Classifier',
                    'Random_Forest_Classifier',
                    'Bagging_Classifier',
                    'AdaBoost_Classifier',
                    'Gradient_Boosting_Classifier',
                    'HistGradient_Boosting_Classifier',
                    'CatBoost_Classifier',
                    'LGBM_Classifier',
                    'XGBoost_Classifier',
                    'Multinomial_NB',
                    'K_Neighbors_Classifier',

               ]
```

In [6]:

```
def train_eval_models(data, classifiers, names):

    """
    data : the dataset with labels and features, not splitted
    classifiers : list of models with parameters
    returns: log_loss comparision of all models
    """

    X_train, X_test, y_train, y_test = train_test_split(data.iloc[:,1:], data['Class'],\
                                                        test_size = 0.4,random_state =123)

    model_losses = []
#   names = []
    cnt = 0

    for mod in classifiers:
        print(f"Running {names[cnt]}....")
        try:
            mod.fit(X_train, y_train)
            y_pred = mod.predict_proba(X_test)
            loss = log_loss(y_test, y_pred,\
                    labels = mod.classes_, eps=1e-15)
#             names.append(str(mod)[:str(mod).index('(')])
            model_losses.append(loss)
        except:
            print(f"Problem in running {names[cnt]}, below is the traceback:")
            print(traceback.print_exc())
        cnt += 1

    clf_compare_dict = dict(zip(names, model_losses))

    clf_compare_df = pd.DataFrame([clf_compare_dict]).T.sort_values(by = 0).\
                                                    rename(columns = { 0 : 'log_loss'})

    return clf_compare_df
```

In [7]:

```
clf_compare = train_eval_models(data = cancer_df, classifiers = classifiers, names = names)
```

```
Running Linear SVM....
Running Polynomial SVM....
Running RBF SVM....
Running Logistic_Regression....

/home/maheswari21303/.local/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.ht
ml)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modul
es/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

Running Extra_Trees_Classifier....
Running Random_Forest_Classifier....
Running Bagging Classifier....
```

In [44]:

```
clf_compare = clf_compare.reset_index().rename(columns = {'index' : 'raw_untuned_models'})
clf_compare
```

Out[44]:

	raw_untuned_models	log_loss
0	XGBoost_Classifier	0.494166
1	LGBM_Classifier	0.521928
2	HistGradient_Boosting_Classifier	0.527336
3	CatBoost_Classifier	0.640087
4	Gradient_Boosting_Classifier	0.668631
5	RBF SVM	0.677973
6	Linear SVM	0.775565
7	Logistic_Regression	0.928543
8	Random_Forest_Classifier	0.932031
9	Polynomial SVM	1.125201
10	Multinomial_NB	1.150596
11	Extra_Trees_Classifier	1.486181
12	AdaBoost_Classifier	1.980917
13	Bagging_Classifier	2.147479
14	K_Neighbors_Classifier	5.255292

In [9]:

```
######################## Train & Test Split ######################################################################
## Train and Test Splitting

X = cancer_df.iloc[:,1:3374]
y = cancer_df.iloc[:,0]

# X = cancer_df.sample(frac=1)[:500]
# y = X.pop('Class')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

In [10]:

```
#################### Log_Loss for GridSearch #################
LogLoss = make_scorer(log_loss, greater_is_better=False, needs_proba=True)
```

In [ ]:

Writing the function for hyperparametric tuning of models

In [11]:

```
def tune_models(estimator, param_grid, X_train, X_test, y_train, y_test, return_model = False):

    """
    estimator: model that has to be tunned
    param_grid: the dictionary of parameters that has to be searched
    X_train, X_test, y_train, y_test
    return_model: turn it to true if you want to return the grid_model object
    """

    gscv = GridSearchCV(estimator = estimator,
                        param_grid = param_grid,
                        scoring = LogLoss,
                        cv = 5, refit = True,
                        n_jobs = -1)

    grid_model = gscv.fit(X_train, y_train)

    print(f'The best parameters are: {grid_model.best_params_}')
    print(" ")

    best_model = grid_model.best_estimator_


    tuned_log_loss_test = log_loss(y_test, best_model.predict_proba(X_test),\
                                   labels = best_model.classes_, eps=1e-15)

    tuned_log_loss_train = log_loss(y_train, best_model.predict_proba(X_train),\
                                    labels = best_model.classes_,eps=1e-15)

    print("Log_loss with tuned parameter on test set is: ", tuned_log_loss_test)
    print("Log_loss with tuned parameter on training set is: ", tuned_log_loss_train)


    if return_model == True:
        return tuned_log_loss_test, tuned_log_loss_train, grid_model
    elif return_model == False:
        return tuned_log_loss_test, tuned_log_loss_train
```

1. Logistic Regression

In [12]:

```
model1 = LogisticRegression(random_state=123, penalty='l2', solver='lbfgs', max_iter=500)

param_grid_model1 = [{'C': [0.1, 0.5, 1.0, 10.0]}]

tuned_log_loss_test_model1, tuned_log_loss_train_model1  = tune_models(

            estimator = model1,
            param_grid = param_grid_model1,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
The best parameters are: {'C': 10.0}

Log_loss with tuned parameter on test set is:  0.741793170114506
Log_loss with tuned parameter on training set is:  0.30521245813407694
```

2. SVM

In [13]:

```
## >>>>>>>>>>>>>>>>> Parameter tuning for SVM <<<<<<<<<<<<<<<<<<<<<< ##

model2 = SVC(probability = True)

param_grid_model2 = {'C': [0.1, 1, 10, 100],
              'gamma': [10, 1, 0.1, 0.01],
              'kernel': ['rbf','poly','linear']}


tuned_log_loss_test_model2, tuned_log_loss_train_model2  = \
tune_models(
            estimator = model2,
            param_grid = param_grid_model2,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
The best parameters are: {'C': 100, 'gamma': 1, 'kernel': 'rbf'}

Log_loss with tuned parameter on test set is:  0.5170206019461544
Log_loss with tuned parameter on training set is:  0.09076392169281747
```

3. Decision Trees

In [14]:

```
############### Decision Tree with tuned parameters ######

model3 = DecisionTreeClassifier()

param_grid_model3 = {"criterion":["gini","entropy"],"splitter":["best","random"],\
              "max_depth":[20,30,50,60,70],"min_samples_split":[2,3,4,5,6]}


tuned_log_loss_test_model3, tuned_log_loss_train_model3  = \
tune_models(
            estimator = model3,
            param_grid = param_grid_model3,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
The best parameters are: {'criterion': 'gini', 'max_depth': 20, 'min_samples_split': 5, 'splitter': 'random'}

Log_loss with tuned parameter on test set is:  3.9545216187909777
Log_loss with tuned parameter on training set is:  0.3362443499429704
```

Since DT log loss is exceptionally high, we will ignore it in comparision

4. Random Forest

In [15]:

```
## Random forest with Tuned Parameters

param_grid = {'n_estimators': [70, 100, 200, 300],'max_depth':[20,30,50,80,100],\
              'min_samples_leaf': [3, 4, 5],'min_samples_split': [8, 10, 12]}

############### Decision Tree with tuned parameters ######

model4 = RandomForestClassifier()

param_grid_model4 =  {'n_estimators': [70, 100, 200, 300],'max_depth':[20,30,50,80,100],\
              'min_samples_leaf': [3, 4, 5],'min_samples_split': [8, 10, 12]}


tuned_log_loss_test_model4, tuned_log_loss_train_model4  = \
tune_models(
            estimator = model4,
            param_grid = param_grid_model4,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
The best parameters are: {'max_depth': 100, 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 100}

Log_loss with tuned parameter on test set is:  1.034071084420567
Log_loss with tuned parameter on training set is:  0.8823464711039153
```

5. Bagging with RF

In [16]:

```
######################################## Bagging Classifier ##############################################################################
##########################################################################################################################################

## Bagging with RF

model5 = BaggingClassifier(base_estimator = RandomForestClassifier(n_estimators = 200))

param_grid_model5 =  {'n_estimators': [10, 20, 30],'bootstrap_features':[True, False],\
                      'max_features' : [1, X_train.shape[1]//2, X_train.shape[1]//4, X_train.shape[1]//8]}


tuned_log_loss_test_model5, tuned_log_loss_train_model5  = \
tune_models(
            estimator = model5,
            param_grid = param_grid_model5,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
/opt/anaconda3/lib/python3.9/site-packages/joblib/externals/loky/process_executor.py:702: UserWarning: A worker stopped whi
le some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(

The best parameters are: {'bootstrap_features': False, 'max_features': 1686, 'n_estimators': 30}

Log_loss with tuned parameter on test set is:  0.6590109150518442
Log_loss with tuned parameter on training set is:  0.2909419916674942
```

6. HistGradientBoosting

In [17]:

```
######################################## Boosting Classifier ##############################################################################
##########################################################################################################################################
 ## Boosting with HistGB

model6 = HistGradientBoostingClassifier()
param_grid_model6 = {
    "max_leaf_nodes" : [28,31,35],
    "max_depth" : [5,6,7,8,9],
    "l2_regularization" : [0, 0.1],
}

tuned_log_loss_test_model6, tuned_log_loss_train_model6  = \
tune_models(
            estimator = model6,
            param_grid = param_grid_model6,
            X_train = X_train, X_test = X_test, y_train = y_train, y_test = y_test
            )
```

```
The best parameters are: {'l2_regularization': 0, 'max_depth': 6, 'max_leaf_nodes': 31}

Log_loss with tuned parameter on test set is:  0.4294029925016674
Log_loss with tuned parameter on training set is:  0.14781639640899186
```

Final comparision

In [58]:

```
log_loss_train = [tuned_log_loss_train_model1,
           tuned_log_loss_train_model2,
#            tuned_log_loss_train_model3,
           tuned_log_loss_train_model4,
           tuned_log_loss_train_model5,
           tuned_log_loss_train_model6]

log_loss_test = [tuned_log_loss_test_model1,
           tuned_log_loss_test_model2,
#            tuned_log_loss_test_model3,
           tuned_log_loss_test_model4,
           tuned_log_loss_test_model5,
           tuned_log_loss_test_model6]

models = ['Logistic Regression',
          'SVM',
#           'Decision Tree',
          'Random Forest',
          'Bagging with Random Forest',
          'Histogram GradientBoosting']
```

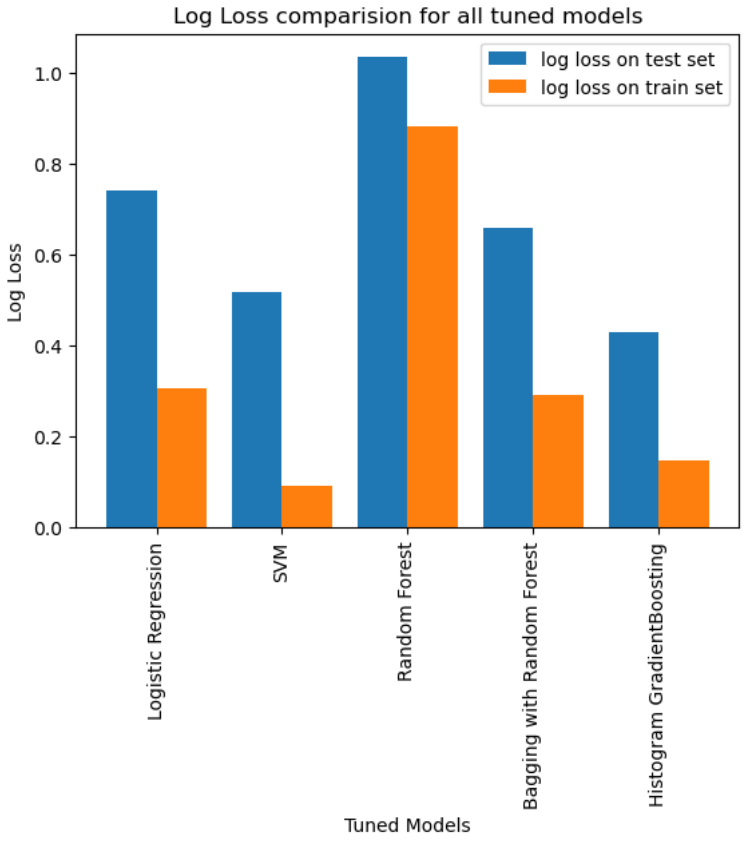
In [59]:

```
X = models
Y_log_loss_test = log_loss_test
Z_log_loss_train = log_loss_train

X_axis = np.arange(len(X))

plt.bar(X_axis - 0.2, Y_log_loss_test, 0.4, label = 'log loss on test set')
plt.bar(X_axis + 0.2, Z_log_loss_train, 0.4, label = 'log loss on train set')

plt.xticks(X_axis, X, rotation = 90)
plt.xlabel("Tuned Models")
plt.ylabel("Log Loss")
plt.title("Log Loss comparision for all tuned models")
plt.legend()
plt.show()
```



In [65]:

```
final_compare.sort_values(by = 'test_logloss')
```

Out[65]:

	models	test_logloss	train logloss
4	Histogram GradientBoosting	0.429403	0.147816
1	SVM	0.517021	0.090764
3	Bagging with Random Forest	0.659011	0.290942
0	Logistic Regression	0.741793	0.305212
2	Random Forest	1.034071	0.882346

The best performing model in this case is Histogram GradientBoosting

In [57]:

```
##############################################################################################
```